Conventional Commit Cheat Sheet

A simple guide to help you understand and apply the Conventional Commit standard for versioning in your projects.

Helps You Be a Better Developer

- Adopting Conventional Commits improves your communication skills by encouraging clear and organized commit messages.
- It helps you focus on your changes and their impact, making it easier to manage projects and collaborate effectively.

How to Use Conventional Commits in Your Git Workflow

1. Commit Messages Using git commit in the Terminal

- When you make changes to your code and want to commit them using **Conventional Commits**, you'll use the git commit command in your terminal. The key is to follow the **Conventional Commit** format for your commit messages.

Example Command:

- In your terminal, run the following:

  ```
  git commit -m "feat(auth): add Google login feature"
  ```

Steps to Commit in the Terminal

1. **Make Changes**: Modify your files as needed.
2. **Stage Your Changes**
   : Add your modified files to the staging area.
   - git add <file>
   - Or to add all changed files at once: git add .

3. **Commit with Conventional Commit Message**
   : After staging the changes, use the following command to commit:
   - git commit -m "feat(button): add rounded corners"

4. **Push the Changes**
   : Push your commits to the remote repository.
   - git push
   - Or if you are pushing to a specific branch: git push origin <branch-name>

Basic Structure

Each commit message follows this structure:

- **type**: Describes the change (e.g., feat, fix, chore)
- **scope**: Optional. Refers to the area of the project being affected (e.g., api, frontend)
- **description**: A short description of the change.

Types of Commit

1. **feat**: A new feature for the user or system
   Example: feat(auth): add Google login feature
2. **fix**: A bug fix for the user or system
   Example: fix(button): resolve issue with button hover state
3. **chore**: Routine tasks like maintenance or updating dependencies
   Example: chore(deps): update react to version 17.0.2
4. **docs**: Documentation updates
   Example: docs(readme): update installation instructions
5. **style**: Changes related to code style (e.g., formatting, missing semi-colons)
   Example: style(button): fix button alignment in CSS
6. **refactor**: Code change that neither fixes a bug nor adds a feature
   Example: refactor(auth): simplify login form validation logic
7. **test**: Adding or updating tests
   Example: test(auth): add unit tests for login function
8. **build**: Changes that affect the build system or external dependencies
   Example: build(webpack): add webpack config for production build
9. **ci**: Continuous integration-related changes
   Example: ci(gitlab): update CI config for deployment pipeline

Learn More

For a deeper understanding of Conventional Commits, check out the official documentation: Conventional Commits.

Tips

- Keep your messages clear and concise.
- Use the type that best represents the change you made.